Lenguajes Formales y Computabilidad
Resumen Teorico
Parcial 2
Guias 4 a 5

Agustín M. Domínguez

1C 2025

Índice

Este apunte tiene las definiciones importantes de la materia, sin varias de las explicaciones o demostraciones que están en las guías, a modo de *resumen*.

Esta materia tiene la particularidad que (aparte de las clases mismas) las guías son el recurso más completo para el estudiante, que funcionan como **Resumen de la clase**, **Apunte Teórico**, y **Práctico de la materia** en un solo documento. Otra particularidad es que las guías son documentos vivos, en el sentido que los profes están constantemente expandiendo y ajustando el contenido, por lo que este documento *puede* estar desactualizado si se consulta en el futuro. Siempre conviene buscar la última versión de las guías y estudiar desde ahí. Sin embargo, este documento puede servir para tener las definiciones importantes de la materia a mano.

Guía 4: El Paragidma de Turing

1.1 Definiciones Intuitivas

El paradigma de Turing es una de las 3 formas de formalizar matemáticamente el concepto de función Γ-Efectivamente computable que vamos a ver en la materia.

> *Def:* **Funciones Γ-Turing (Versión intuitiva)**
> Son las funciones que pueden ser computadas por una *máquina de Turing*

La máquinas de Turing son un modelo abstracto de máquina que computa cosas.

> *Def:* **Máquina de Turing (Versión intuitiva)**
> El ejemplo más simple y concreto de una máquina de Turing es:
> Dado un alfabeto Γ, es un dispositivo que tiene:
> - Una cantidad **finita** de estados.
> - Una cinta de papel infinita dividida en cuadros, los cuales pueden guardar símbolos de Γ o estar vacíos.
> - El cabezal inicia en el segundo cuadrp de la cinta (es decir con 1 espacio a la izquierda e infinitos a la derecha)
> - Un cabezal que puede *'trabajar'* sobre un cuadro a la vez.
> - El cabezal puede *'trabajar'* sobre el cuadro de las siguientes formas:
>   Leer el simbolo del recuadro
>   Moverse un lugar a la izquierda o derecha
>   Escribir en el cuadro cualquier símbolo de Γ
>   Borrar el símbolo actualmente escrito en el cuadro
> - Por último el comportamiento del cabezal depende del estado de la máquina, la cual puede cambiar al *'trabajar'* sobre un cuadro.
> - La cinta puede tener cualquer combinacion de símbolos de Γ en la cinta, la cual considerarmos el *input* del programa

> *Def:* **B**
> En general en lugar de pensar que un cuadro está 'en blanco', asumiremos que uno de los símbolos de Γ significa esto, el cual por convención lo nombraremos como **B**

Def: **Q y q_0**

También por convención llamaremos **Q** al conjunto de estados de la máquina
También toda máquina deberá tener un estado inicial el cual denotaremos como q_0

Def: **Detención de la Máquina de Turing**

Diremos que una máquina de Turing se detiene cuando:

- La máquina no tiene contemplada una acción para cierto estado
- La máquina intente moverse a la izquierda cuando ya está en el primer lugar

Def: **Alfabeto de entrada**

Al siguiente alfabeto lo denotaremos con Σ y lo llamaremos alfabeto de entrada.
Es un alfabeto $\Sigma \subseteq \Gamma$ formado por los símbolos que permitimos que la cinta esté en su configuración inicial al empezar el programa, con excepción del símbolo B.
Es decir que cada símbolo de la configuración de la cinta inicial tiene que ser B o pertenecer a Σ.
Los símbolos de $\Gamma - \Sigma$ los consideramos *auxiliares*

Def: **Estados Finales**

Dado Q los estados de una máquina, definiremos un subconjunto de estos estados.
Lo denotaremos con F y los llamaremos estados *finales*.

Nota: A pesar de la primera intuición que tenemos ante la mención de 'finales', notar que la definición estricta no dice nada de qué condiciones debe cumplir los elementos de F. No dice, por ejemplo, que la máquina M debe detenerse en un estado final, o que cuando llega a un estado final se detiene. Tampoco dice cuantos debe haber o si deben existir. En resumen, son arbitrariamente elegidos para propósitos que tendrán más sentido cuando veamos la definición de Lenguaje.

Def: **Palabra Aceptada por M por alcance de estado final**

Dado una máquina de Turing M, y una palabra $\alpha = a_1 \dots a_n \in \Sigma^*$
Diremos que α es aceptada por M por alcance de estado final si partiendo de la cinta:

$$\overset{(q_0)}{B} \quad a_1 \quad \dots \quad a_n \quad B \quad B \dots$$

En algun momento de la computación, la máquina M está en un estado de F

Def: **Lenguaje de una máquina de Turing**

Dado una máquina de Turing M, definimos el lenguaje de M, denotado como $L(M)$ al conjunto de formado por todas las palabras que son aceptadas por alcance de estado final.

Intuitivamente, son las palabras que si las ponemos como input de la máquina, terminan en un estado final de F

1.2 Definiciones Matemáticas

Def: **Máquina de Turing (Versión Matemática)**

Una máquina de Turing es una 7-upla $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ donde

- Q es un conjunto finito cutos elementos son llamados *estados*
- Γ es un alfabeto
- Σ es un *alfabeto de entrada* tal que $\Sigma \subseteq \Gamma$
- $B \in \Gamma - \Sigma$ es un símbolo llamado el *blank symbol*
- $\delta : D_\delta \subseteq Q \times \Gamma \to Q \times \Gamma \times \{L, R, K\}$ es una función que define el **comportamiento** de la máquina
- q_0 es un estado llamado el *estado inicial* de M
- $F \subseteq Q$ es un conjunto de estados llamados *finales*

Los símbolos L, R, K serviran para especificar que hará el cabezal:

- L = Left (moverse un lugar a la izquierda)
- R = Right (moverse un lugar a la derecha)
- K = Keep (quedarse quieto)

La función σ define el comportamiento (o 'programación') de la máquina:

$$\sigma(A, B) = (C, D, E)$$

Donde

- A = Estado actual
- B = Símbolo que el cabezal lee
- C = Estado al que pasa la máquina
- D = Símbolo que se escribe en el espacio actual
- E = Movimiento que realiza el cabezal

Def: **Descripciones Instantáneas**

Es una forma eficiente de denotar el estado actual de una Máquina de Turing, define el contenido de la cinta, el lugar donde esta el cabezal, y el estado actual.
Una descripción instantánea será una palabra de la forma:

$$\alpha q \beta$$

Donde

- $\alpha, \beta \in \Gamma^*$
- $q \in Q$
- $\beta_i \neq B$, donde $i = |\beta|$ (es decir $\beta = \varepsilon$ o su último símbolo no es B)

Representará la situación:

$$\alpha_1 \quad \alpha_2 \quad \dots \quad \alpha_n \quad \underset{q}{\beta_1} \quad \beta_2 \quad \dots \quad \beta_m \quad B \quad B \dots$$

Notar que si $n = 0$ entonces el estado es:

$$\underset{q}{\beta_1} \quad \beta_2 \quad \dots \quad \beta_m \quad B \quad B \dots$$

y si $m = 0$ el estado es:

$$\alpha_1 \quad \alpha_2 \quad \dots \quad \alpha_n \quad \underset{q}{B} \quad B \dots$$

Def: **Des**

Usaremos *Des* para denotar el conjunto de las descripciones instantáneas.

Def: **Función St**

$$\begin{aligned} St : Des &\to Q \\ d &\to \text{único símbolo de Q que occurre en } d \end{aligned} \tag{1.1}$$

En palabras es la función que dada una descripción instantánea $\alpha q \beta$ devuelve q, o dicho de otra manera toma una descripción instantánea y devuelve el estado que esta descripción enuncia

Def: **Función $\lfloor\rfloor$**

Dado $\alpha \in (Q \bigcup \Gamma)^*$ y $\sigma \in Q \bigcup \Gamma$, definamos $\lfloor\alpha\rfloor$ de la siguiente manera:

$$\lfloor\varepsilon\rfloor = \varepsilon$$
$$\lfloor\alpha B\rfloor = \lfloor\alpha\rfloor \quad \text{si } \sigma \neq B$$
$$\lfloor\alpha\sigma\rfloor = \alpha\sigma$$

Es decir el resultado de remover de α el tramo final más grande de la forma B^n

Def: **Función $\rightsquigarrow$ y $\hookleftarrow$**

Dado α palabra:

$$\rightsquigarrow \alpha = \begin{cases} [\alpha]_2 \cdots [\alpha]_{|\alpha|} & \text{si } |\alpha| \geq 2 \\ \varepsilon \quad \text{si } |\alpha| \leq 1 \end{cases}$$

$$\alpha \hookleftarrow = \begin{cases} [\alpha]_1 \cdots [\alpha]_{|\alpha-1|} & \text{si } |\alpha| \geq 2 \\ \varepsilon \quad \text{si } |\alpha| \leq 1 \end{cases}$$

Def: **Relación** $\vdash$

Dadas $d_1, d_2 \in Des$, escribiremos $d_1 \vdash d_2$ cuando existan $\sigma \in \Gamma$, $\alpha, \beta \in \Gamma^*$ y $p, q \in Q$ tales que $d_1 = \alpha p \beta$ y se cumpla alguno de los siguientes casos:

Caso 1:

$$\delta(p, [\beta B]_1) = (q, \sigma, R)$$

$$d_2 = \alpha \sigma q (\rightsquigarrow \beta)$$

Caso 2:

$$\delta(p, [\beta B]_1) = (q, \sigma, L) \text{ y } \alpha \neq \varepsilon$$

$$d_2 = \lfloor (\alpha \leftsquigarrow) q [\alpha]_{|\alpha|} \sigma (\rightsquigarrow \beta) \rfloor$$

Caso 3:

$$\delta(p, [\beta B]_1) = (q, \sigma, K)$$

$$d_2 = \lfloor \alpha q \sigma (\rightsquigarrow \beta) \rfloor$$

Esto es un montón de notación para decir en palabras lo siguiente:
$d_1 \vdash d_2$ dice que d_1 es una descripción instantánea de una máquina de Turing M, y d_2 es la descripción instantánea de M al **siguiente** paso. Es decir al ejecutar el siguiente paso de la máquina descrita por d_1, obtengo la máquina descrita por d_2

Def: $\nvdash$

Escribiremos $d_1 \nvdash d_2$ para expresar que no se da $d_1 \vdash d_2$

Def: $\overset{n}{\vdash}$

Dados $d, d' \in Des \wedge n \geq 0$ escribiremos $d \overset{n}{\vdash} d'$ si existen $d_1, \ldots, d_n$ tal que:

$$d = d_1 \vdash d_2 \vdash \cdots \vdash d_n = d'$$

Notar que: $d \overset{0}{\vdash} d' \iff d = d'$

Def: $\overset{*}{\vdash}$

$$d \overset{*}{\vdash} d' \iff \exists\, n \in \omega \mid d \overset{n}{\vdash} d'$$

Def: **Detención (Versión Matemática)**

Dada $d \in Des$, diremos que M se detiene partiendo de d su existe $d' \in Des$ tal que:

- $d \overset{*}{\vdash} d'$
- $d' \nvdash d''$ para cada $d'' \in Des$

En palabras es cuando hay un 'camino' entre d y d', y no hay caminos que salen de d'

Si bien no está explícito, esta definición también cubre el caso donde la máquina se detiene por intentar ir a la izquierda en la primera celda de la cinta. Esto es porque si estamos en d' donde el estado está en el primer lugar de la cinta y la acción correspondiente de δ incluye ir a la izquierda, entonces la máquina se detiene (igual que en la descrición intuitiva). Y como se detiene, entonces no existe d'' tal que $d' \vdash d''$

Def: **Lenguaje $L(M)$ (Versión Matemática)**

Diremos que una palabra $\alpha \in \Sigma^*$ es aceptada por M por alcance de estado final cuando

$$\lfloor q_0 B \alpha \rfloor \overset{*}{\vdash} d, \text{ con } d \text{ tal que } St(d) \in F$$

Luego
$L(M) = \{\alpha \in \Sigma^* \mid \alpha \text{ es aceptada por M por alcance de estado final}\}$
Nota: es la misma idea que la definición intuitiva pero más formal. Dado un conjunto de estados F a los que llamamos finales, si hay un camino del estado inicial de la máquina a ese estado, entonces cuenta como parte del lenguaje

1.3 Funciones Σ-Turing Computables

Def: **Símbolo *unit***

Para poder computar funciones mixtas con una maquina de Turing necesitaremos un simbolo para representar numeros sobre la cinta. Llamaremos a este simbolo unit y lo denotaremos con |

¿Como vamos a usar *unit* para representar números?: Vamos a usar el símbolo | para expresar números en la cinta de Turing con un **sistema unario**, es decir de base 1, donde por ejemplo | representa 1, || representa 2, ||||| representa 5, etc.

Def: **Máquina de Turing con unit**

Es una 8-tupla $M = (Q, \Sigma, \Gamma, \delta, q_0, B, |, F)$
donde $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$ es una máquina de Turing y el símbolo $| \in \Gamma - (\{B\} \cup \Sigma)$

Def: **Función Σ-Turing Computable (Versión palabra)**

Una función $f : D_f \subseteq \omega^n \times \Sigma^{*m} \to \Sigma^*$ es Σ-Turing computable si existe una maquina de turing con unit $M = (Q, \Sigma, \Gamma, \delta, q_0, B, |, F)$ tal que:
(1) Si $(\overrightarrow{x}, \overrightarrow{\alpha}) \in D_f$, entonces hay un estado $p \in Q$ tal que

$$\lfloor q_0 B \,|^{x_1}\, B \,|^{x_2}\, B \ldots |^{x_m}\, B\ \alpha_1\ B\ \alpha_2\ B\ \ldots\ B\ \alpha_m \rfloor \overset{*}{\vdash} \lfloor p\ B f(\overrightarrow{x}, \overrightarrow{\alpha}) \rfloor$$

y

$$\lfloor p\ B\ f(\overrightarrow{x}, \overrightarrow{\alpha}) \rfloor \nvdash d\ \forall d \in Des$$

(2) Si $(\overrightarrow{x}, \overrightarrow{\alpha}) \in \omega^n \times \Sigma^{*m} - D_f$ (no está en el dominio de D_f), entonces M **no** se detiene partiendo de $\lfloor q_0 B \,|^{x_1}\, B \,|^{x_2}\, B \ldots |^{x_m}\, B\ \alpha_1\ B\ \alpha_2\ B\ \ldots\ B\ \alpha_m \rfloor$

Def: **Función Σ-Turing Computable (Versión número)**

Una función $f : D_f \subseteq \omega^n \times \Sigma^{*m} \to \Sigma^*$ es Σ-Turing computable si existe una maquina de turing con unit $M = (Q, \Sigma, \Gamma, \delta, q_0, B, |, F)$ tal que:
(1) Si $(\overrightarrow{x}, \overrightarrow{\alpha}) \in D_f$, entonces hay un estado $p \in Q$ tal que

$$\lfloor q_0\ B \,|^{x_1}\, B \,|^{x_2}\, B \ldots |^{x_m}\, B\ \alpha_1\ B\ \alpha_2\ B\ \ldots\ B\ \alpha_m \rfloor \overset{*}{\vdash} \lfloor p\ B \,|^{f(\overrightarrow{x}, \overrightarrow{\alpha})} \rfloor$$

y

$$\lfloor p\ B \,|^{f(\overrightarrow{x}, \overrightarrow{\alpha})} \rfloor \nvdash d\ \forall d \in Des$$

(2) Si $(\overrightarrow{x}, \overrightarrow{\alpha}) \in \omega^n \times \Sigma^{*m} - D_f$ (no está en el dominio de D_f), entonces M **no** se detiene partiendo de $\lfloor q_0\ B \,|^{x_1}\, B \,|^{x_2}\, B \ldots |^{x_m}\, B\ \alpha_1\ B\ \alpha_2\ B\ \ldots\ B\ \alpha_m \rfloor$

Muchos símbolos, pero en palabras, ambas definiciones son 'traducciones' de la definición de proceso efectivo computable a la máquina de Turing.

Empecemos con el estado inicial de la máquina de Turing: '$\ldots q_0 B \ldots$' describe el estado inicial de la máquina de Turing, con un espacio en blanco al principio. Esto va a estar seguido del input de la función.

Acá es donde viene primera traducción: ¿Como se escribe el input de la función (que es una (n+m)-tupla $\in \omega^n \times \Sigma^{*m}$)?

El input tiene una parte en números, y una parte en palabras de Σ^*. La parte de números la escribimos usando el sistema en base 1 con '|', y las palabras las escribimos directamente en la descripción instantanea. Para distinguir cuando termina un número o palabra y empieza el siguiente, separamos ambos elementos con una B.

Veamos un ejemplo: Si suponemos $\Sigma = \{\%, @\}$ y tenemos una función con $D_f \subseteq w^2 \times \Sigma^{*2}$ y el input es $(2, 3, \%@\%, @@@)$, en la descripción instantanea del estado inicial de M la cinta se vera como:

$$q_0\ B\ |\ |\ B\ |\ |\ |\ B\ \%\ @\ \%\ B\ @\ @\ @$$

Luego de todo eso tenemos "$\ldots \overset{*}{\vdash} \lfloor p\ Bf(\overrightarrow{x}, \overrightarrow{\alpha})\rfloor$" que es la segunda traducción. Esto dice que el estado inicial de M eventualmente avanza hasta un estado p y en la cinta esta escrito la representación de Turing del valor de $f(\overrightarrow{x}, \overrightarrow{\alpha})$, es decir que la máquina 'calculó' $f(\overrightarrow{x}, \overrightarrow{\alpha})$ y lo en la cinta.

Para la segunda definición, lo único que cambia es el estado final, ya que la función devuelve algo de ω en lugar de algo de Σ^*. Entonces, como en la primera definición devuelve $f(\overrightarrow{x}, \overrightarrow{\alpha})$ escrito como '$\lfloor p\ Bf(\overrightarrow{x}, \overrightarrow{\alpha})\rfloor$', en este caso devuelve también $f(\overrightarrow{x}, \overrightarrow{\alpha})$, escrito como '$\lfloor p\ B\ |^{f(\overrightarrow{x}, \overrightarrow{\alpha})}\rfloor$'

Todo eso es el caso **(1)** donde $(\overrightarrow{x}, \overrightarrow{\alpha})$ está en el dominio de la función. El caso **(2)** simplemente dice que no se detiene a partir del estado inicial de M. Recordemos que q_0 se define como el estado inicial, y la entrada de la funciòn se escribe al principio de la cinta luego de escribir un B, por lo que $\lfloor q_0\ B\ |^{x_1}\ B\ |^{x_2}\ B \ldots |^{x_m}\ B\ \alpha_1\ B\ \alpha_2\ B\ \ldots\ B\ \alpha_m\rfloor$ describe ese estado inicial.

Proposición: **Leibniz vence a Turing**

Si $f : D_f \subseteq \omega^n \times {\Sigma^*}^m \to O$ con $O \in \{\omega, \Sigma^*\}$ es computada por una máquina de turing con unit $M = (Q, \Sigma, \Gamma, \delta, q_0, B, |, F)$, entonces f es Σ-efectivamente computable

Demonstración en la guía.

Guía 5: El Paradigma de Gödel (Parte 1)

2.1 PR_0 y Composición

En esta parte vemos el paradigma de **Funciones Σ-Recursivas Primitivas**, dado por Gödel

La idea principal de Gödeles partir de un conjunto muy limitado de funciones que está demostrado son Σ-Efectivamente Computables, y a partir de ellas nuevas funciones preservando su "computabilidad". Para esto se apoya en **constructores** que son formas de combinar funciones para crear nuevas funciones con la importante característica que si las funciones usadas ya son computables, entonces la función resultante también lo es (es decir que los constructores preservan computabilidad)

Def: **Familia simple, o PR_0 (primera capa de funciones Primitivamente Recursivas)**
Estas funciones que son evidentemente Σ-Computables es la siguiente familia:

$$\{Suc, Pred, C_0^{0,0}, C_\varepsilon^{0,0}\} \bigcup \{d_a \mid a \in \Sigma\} \bigcup \{p_j^{n,m} \mid 1 \leq j \leq n+m\}$$

Y los constructores que veremos serán:

- Composición
- Recursión Primitiva
- Minimización de predicados totales (esta la vemos en Parte 2)

Def: **Composición**
Dadas funciones Σ-Mixtas $f, f_1, \ldots, f_r$ con $r \geq 1$, diremos que la función

$$f \circ [f_1, \ldots, f_r]$$

es obtenida por composición a partir de las funciones $f, f_1, \ldots, f_r$

Lemma: **La composición es Σ-Mixta**

Dadas funciones Σ-Mixtas $f, f_1, \ldots, f_r$ con $r \geq 1$, tales que $f \circ [f_1, \ldots, f_r] \neq \varnothing$ entonces existen $n, m, k, l \in \omega, s \in \{\#, *\}$ tales que:

- $r = n + m$
- f es detipo (n, m, s)
- f_i es de tipo $(k, l, \#)$, para cada $i = 1, \ldots, n$
- f_i es de tipo $(k, l, *)$, para cada $i = n + 1, \ldots, n + m$

Y más aún:
En este caso la función es de tipo (k, l, s)

Lemma: **Preservación de computabilidad**

Si $f, f_1, \ldots, f_r$ con $r \geq 1$ son Σ-Efectivamente Computables, entonces $f \circ [f_1, \ldots, f_r]$ lo es.

Prueba en la guía.

2.2 Recursión Primitiva

La recursión es un tipo muy particular (y limitado) de recursión que va a ser el que nos enfoquemos en la materia.

La primera limitación que vamos a hacer es que vamos a pensar en la recusión sobre una **única** variable. Este obviamente no es el único tipo de recursión que existe, pero es el que vamos a estudiar.

Como estamos hablando de funciones Σ-Mixtas, estamos trabajando con cosas que tienen números y palabras, entonces podemos hablar de recursión sobre palabras y sobre números. Esto es una de las separaciones. También puede ser que dicha función devuelva cosas de ω o cosas de Σ^*, por lo que son 4 tipos de recursión primitiva que vamos a estudiar:

- Recursión sobre una variable numérica que devuelve números
- Recursión sobre una variable numérica que devuelve palabras
- Recursión sobre una variable alfabética que devuelve números
- Recursión sobre una variable alfabética que devuelve palabras

Def: **Conjunto Rectangular**

Sea Σ un alfabeto finito, un conjunto Σ-Mixto S es llamado *rectangular* si es de la forma:

$$S_1 \times \cdots \times S_n \times L_1 \times \cdots \times L_m$$

con cada $S_i \subseteq \omega$ y cada $L_i \subseteq (\overrightarrow{x}, \overrightarrow{\alpha})$
Por notación, a veces anotaremos $S \times L$ para delimitar las partes de un conjunto rectagular

Lemma:

Sea $S \subseteq \omega \times \Sigma^{m*}$. Entonces

$$S \text{ es rectangular } \iff (x,\alpha),(y,\beta) \in S \implies (x,\beta) \in S$$

Def: **Recursión Primitiva sobre variable numérica con valores numéricos**

Dadas las funciones:

$$f \ : \ S_1 \times \cdots \times S_n \times L_1 \times \cdots \times L_m \to \omega$$

$$g \ : \ \omega \times \omega \times S_1 \times \cdots \times S_n \times L_1 \times \cdots \times L_m \to \omega$$

Con $S_1, \ldots, S_n \subseteq \omega$ y $L_1, \ldots, L_m \subseteq \Sigma^*$ no vacíos.
Llamaremos $R(f,g)$ a la **única** función:

$$R : \omega \times S_1 \times \cdots \times S_n \times L_1 \times \cdots \times L_m \to \omega$$

Tal que:
$R(f,g)(0, \overrightarrow{x}, \overrightarrow{\alpha}) = f(\overrightarrow{x}, \overrightarrow{\alpha})$
$R(f,g)(t+1, \overrightarrow{x}, \overrightarrow{\alpha}) = g(A, t, \overrightarrow{x}, \overrightarrow{\alpha})$
con $A = R(f,g)(t, \overrightarrow{x}, \overrightarrow{\alpha})$
Donde diremos que $R(f,g)$ es obtenida por *recusión primitiva* a partir de f y g.

Tener en cuenta que $R(f,g)$ es notación para llamar a la función. R **no** es una función que tome dos funciones como argumento.

Def: **Recursión Primitiva sobre variable numérica con valores alfabeticos**

Dadas las funciones:

$$f \ : \ S_1 \times \cdots \times S_n \times L_1 \times \cdots \times L_m \to \Sigma^*$$

$$g \ : \ \omega \times S_1 \times \cdots \times S_n \times L_1 \times \cdots \times L_m \times \Sigma^* \to \Sigma^*$$

Con $S_1, \ldots, S_n \subseteq \omega$ y $L_1, \ldots, L_m \subseteq \Sigma^*$ no vacíos.
Llamaremos $R(f,g)$ a la **única** función:

$$R : \omega \times S_1 \times \cdots \times S_n \times L_1 \times \cdots \times L_m \to \Sigma^*$$

Tal que:
$R(f,g)(0, \overrightarrow{x}, \overrightarrow{\alpha}) = f(\overrightarrow{x}, \overrightarrow{\alpha})$
$R(f,g)(t+1, \overrightarrow{x}, \overrightarrow{\alpha}) = g(t, \overrightarrow{x}, \overrightarrow{\alpha}, A)$
con $A = R(f,g)(t, \overrightarrow{x}, \overrightarrow{\alpha})$
Donde diremos que $R(f,g)$ es obtenida por *recusión primitiva* a partir de f y g.

Lemma:

Si f y g son Σ-Efectivamente computables, entonces $R(f,g)$ lo es.

Def: **Familia Σ-Indexada de funciones**

Dado un alfabeto Σ, una Familia Σ-Indexada de funciones será una función $\mathcal{G}$ tal que $D_{\mathcal{G}} = \Sigma$ y para cada $a \in D_{\mathcal{G}}$ se tiene que $\mathcal{G}(a)$ es una funcion.
Por notación escribiremos $\mathcal{G}_a$ en lugar de $\mathcal{G}(a)$

Def: **Recursión Primitiva sobre variable alfabética con valores numéricos**

Dadas las funciones:

$$f \ : \ S_1 \times \cdots \times S_n \times L_1 \times \cdots \times L_m \to \omega$$

y una función $\mathcal{G}$ tal que para cada $a \in \Sigma$:

$$\mathcal{G}_a \ : \ \omega \times S_1 \times \cdots \times S_n \times L_1 \times \cdots \times L_m \times \Sigma^* \to \omega$$

Con $S_1, \ldots, S_n \subseteq \omega$ y $L_1, \ldots, L_m \subseteq \omega$ no vacíos.
Definimos $R(f,\mathcal{G})$ a la **única** función:

$$R : S_1 \times \cdots \times S_n \times L_1 \times \cdots \times L_m \times \Sigma^* \to \omega$$

Tal que:
$R(f,\mathcal{G})(\overrightarrow{x}, \overrightarrow{\alpha}, \varepsilon) = f(\overrightarrow{x}, \overrightarrow{\alpha})$
$R(f,\mathcal{G})(\overrightarrow{x}, \overrightarrow{\alpha}, \alpha a) = g_a(A, \overrightarrow{x}, \overrightarrow{\alpha}, \alpha)$
con $A = R(f,\mathcal{G})(\overrightarrow{x}, \overrightarrow{\alpha}, \alpha)$
Donde diremos que $R(f,\mathcal{G})$ es obtenida por *recusión primitiva* a partir de f y $\mathcal{G}$.

Def: **Recursión Primitiva sobre variable alfabética con valores alfabeticos**
Dadas las funciones:

$$f \ : \ S_1 \times \cdots \times S_n \times L_1 \times \cdots \times L_m \to \Sigma^*$$

y una función $\mathcal{G}$ tal que para cada $a \in \Sigma$:

$$\mathcal{G}_a \ : \ S_1 \times \cdots \times S_n \times L_1 \times \cdots \times L_m \times \Sigma^* \times \Sigma^* \to \Sigma^*$$

Con $S_1, \ldots, S_n \subseteq \omega$ y $L_1, \ldots, L_m \subseteq \Sigma^*$ no vacíos.
Definimos $R(f, \mathcal{G})$ a la **única** función:

$$R : S_1 \times \cdots \times S_n \times L_1 \times \cdots \times L_m \times \Sigma^* \to \Sigma^*$$

Tal que:
$R(f, \mathcal{G})(\overrightarrow{x}, \overrightarrow{\alpha}, \varepsilon) = f(\overrightarrow{x}, \overrightarrow{\alpha})$
$R(f, \mathcal{G})(\overrightarrow{x}, \overrightarrow{\alpha}, \alpha a) = g_a(\overrightarrow{x}, \overrightarrow{\alpha}, \alpha, A)$
con $A = R(f, \mathcal{G})(\overrightarrow{x}, \overrightarrow{\alpha}, \alpha)$
Donde diremos que $R(f, \mathcal{G})$ es obtenida por *recusión primitiva* a partir de f y $\mathcal{G}$.

Lemma:
Si f $\mathcal{G}_a$ para cada $a \in \Sigma$ son Σ-Efectivamente computables, entonces $R(f, \mathcal{G})$ lo es.

2.2.1 Proceso recomendado para construir recursiones primitivas.

En varios ejercicios de las guías hay que expresar funciones como recursiones (es decir con notación $R(f, g)$), o lo opuesto que es interpretar funciones ya en formato $R(f, g)$ como las funciones tradicionales que estamos más acostumbrados.

Para esto es importante entender bastante bien el lenguaje de $R(f, g)$ y como convertir una función a ese formato. En esta parte, explicamos los pasos que se recomiendan para que este proceso anti-intuitivo sea más realizable.

Tenemos una función, generalmente expresada en formato lambda. En esta sección hacemos un ejemplo 'fácil' ($\lambda xy[x + y]$) y luego otro más dificil para mostrar el método.

Paso 1: *Identificar la variable recursiva.*

En las recursiones que convertimos en $R(f, g)$ solo hay **una** variable recursiva, que es la que se 'reduce' en llamadas recursivas, y es la que tiene como valor 0 o ε en el caso base. Entonces es importante identificar esta variable para ver en cual de los 4 casos estamos:

- NN: Recursión sobre una variable numérica que devuelve números
- NA: Recursión sobre una variable numérica que devuelve palabras
- AN: Recursión sobre una variable alfabética que devuelve números
- AA: Recursión sobre una variable alfabética que devuelve palabras

La idea general es que la variable numérica tiene que estar en un extremo de los argumentos, primera si es una variable numérica, y última si no es numérica (la definición de $R(f, g)$ asume esto). En casos donde preferiríamos hacer recursion sobre una variable que no cumple esto, hay algunos trucos para convertirla en otra función que si cumpla esto, pero esto ya es un truco más 'avanzado'.

En nuestro ejemplo, $\lambda xy[x + y]$, como son variables numéricas y la suma devuelve numéros, estamos en el primer caso, el caso **NN**, por lo que la variable recursiva debe ser la primera, en este caso x

Paso 2: *Reescribir la función como ecuaciones recursivas.*

Una vez que tenemos la variable, para tener una mejor idea de como vamos a plantear la recursión, nos conviene escribir las ecuaciones recursivas. Esta siempre tiene un caso base.

En los casos donde la variable recursiva es numérica (**NN** y **NA**), el caso base es como se evalua cuando la variable recursiva es 0, y en los casos donde la variable recursiva es alfabética (**AN** y **AA**), el caso base es como se evalua la función cuando la variable recursiva es ε.

Luego el caso recursivo es como calculamos el 'sucesor' a partir del resultado actual.

Cuando la variable recursiva es numérica esto es $t + 1$ y cuando es alfabética es αa con $a \in \Sigma$. Ya veremos las particularidades del caso recursivo con variable alfabética pero veamos el ejemplo simple:

Llamo $F = \lambda xy[x + y]$

Anotaremos los pasos intermedios pero en el caso base queremos la expresión más simple:

$$F(0, y) = 0 + y = y$$

y en el caso recursivo queremos expresar el resultado **a partir del valor anterior**. Es decir en este caso queremos expresar $F(t + 1, y)$ usando en algún lugar $F(t, y)$

$$F(t + 1, y) = (t + 1) + y = (t + y) + 1 = F(t, y) + 1$$

entonces nos quedamos con:

- $F(0, y) = y$
- $F(t + 1, y) = F(t, y) + 1$

Paso 3: *Escribir el tipo de f y g.*

Antes de pensar como escribir f y g, conviene escribir el tipo que tiene la función, es decir su dominio y su conjunto de llegada.

El conjunto de llegada de tanto f como g es **igual** al conjunto de llegada de la función original (que ahora estamos llamando F). Entonces en este caso es ω porque F devuelve numéros

El dominio de f siempre es **'uno menos' que el de la funcion original** y el de g siempre es **'uno más' que el de la funcion original**. Con esto nos referimos al ancho del dominio.

La f pierde el elemento de la variable recursiva, es decir que tiene un argumento menos y por lo tanto el ancho del dominio es igual al de F pero con 'uno menos'. Notar que argumento que pierde es el da la variable recursiva, por lo que si estamos en el caso donde la variable recursiva es numérica, pierde el **primer** argumento, mientras que si la variable es alfabética, pierde el **último**

En este caso pierde el primer argumento, por lo que f se declara como:

$f : \omega \to \omega$

g tiene 'uno más' porque tiene todos los argumentos originales de F, pero se le agrega el valor **Anterior**. Si la función devuelve numéros, entonces el anterior también va a ser un número y va a ir principio de todo, y si devuelve palabras, el anterior también lo será y la vamos a mandar al al final de los argumentos.

Entonces g se declara como:

$g : \omega \times \omega \times \omega \to \omega$

El primer elemento es para el anterior, el segundo para la variable recursiva, y la tercera es para lo que llamamos el **Bloque Fijo**. El bloque fijo es una parte $\omega^n \times {\Sigma^*}^m$ **formado por todas las variables de la función original que NO son recursivas**. En este caso el bloque fijo es solamente y porque es la única variable que no es recursiva, pero en otros casos más complejos puede tener más variables, numéricas o alfabéticas. Le decimos 'bloque fijo' porque siempre se mantiene 'al centro' del dominio y los demás objetos se agregan al exterior. Este es el resumen de como se declara f y g a partir del bloque fijo:

$$f : \text{BLK} \to O$$

Es decir la f tiene **solo** bloque fijo. El dominio de la g va a ser de la forma:

$$\begin{array}{lllll} \text{Funcion NN} & g & : & \omega_A \times \omega_R \times \text{BLK} & \\ \text{Funcion NA} & g & : & \omega_R \times \text{BLK} & \times \Sigma_A^* \\ \text{Funcion AN} & g & : & \omega_A \times \text{BLK} & \times \Sigma_R^* \\ \text{Funcion AA} & g & : & \text{BLK} & \times \Sigma_R^* \times \Sigma_A^* \end{array} \tag{2.1}$$

Donde ω_A y Σ_A^* significa que es de tipo ω o Σ^* respectivamente, pero que corresponde al valor **Anterior** en g, y ω_R y Σ_R^* significa que es de tipo ω o Σ^* respectivamente, pero que corresponde al valor de la variable **Recursiva** en g

Notar que O_R (con $O \in \{\omega, \Sigma^*\}$) siempre va 'pegado' al bloque fijo, y O_A siempre va a uno de los extremos. Intuitivamente, siempre van a un lado o del otro del bloque fijo.

En este caso, nos quedamos con la siguiente declaración de f y g:

- $f : \omega \to \omega$
- $g : \omega \times \omega \times \omega \to \omega$

Paso 4: *Definir la f*

Ahora tenemos todo el trabajo hecho para definir las funciones. La más facil casi siempre es la f ya que es un caso base:

En resumen el desafío es definir una función $f : \omega \to \omega$ tal que **valga lo mismo que la primera ecuación del Paso 2**. En este caso dicha ecuación es $F(0, y) = y$

en este caso es facil: $f(y) = y$

En algunos ejercicios esto alcanza, aunque para probar completamente que esto es una recursión Σ-PR, hay que probar que la f es PR, y esto siempre se hace escribiendo la f como composición o recursión de funciones Σ-Pred

Este es un trabajo puramente artesanal que requiere recordar cuales funciones asumimos que son Σ-PR y cuales otras ya probamos que lo son. En este caso es facil ya que $f(y) = y$ también **es** la función $P_1^{1,0}$ que es una de las funciones base que son Σ-PR

Paso 5: *Definir la g o $\mathcal{G}$*

Similar al paso anterior, queremos escribir una función que cumpla la declaración que hicimos, que valga lo mismo que la segunda ecuación del Paso 2, para probar que es Σ-PR, queremos construirla de funciones Σ-Pred

en este caso queremos una $g : \omega \times \omega \times \omega \to \omega$ tal que se 'parezca' a $F(t + 1, y) = F(t, y) + 1$. Notar que $F(t, y)$ es el 'valor anterior' a $F(t + 1, y)$

En este caso:

$g(A, x, y) = A + 1$ ('A' de anterior)

Pero como queremos escribirla con funciones Σ-PR, hay una función que ya cumple ese "+1"

$g(A, x, y) = Suc(A)$

Notar también que hay variables de g que **no** usamos, en este caso x e y. Esto es normal y no hay problema con no usar todos los argumentos.

Si la variable recursiva es alfabética en lugar de numérica, entonces en lugar de g hay que definir una $\mathcal{G}$, que es, hablando mal y pronto, una g para cada tipo de sucesor posible en una palabra. Si tenemos un lenguaje $\Sigma = \{\$, \%\}$, entonces a una palabra $\alpha \in \Sigma^*$, decimos que tiene dos posibles sucesores: $\alpha\$$ y $\alpha\%$. La idea es la misma pero nos permite definir distintos comportamientos dependiendo del caracter que este siendo el sucesor.

En resumen, los pasos son:

1. **Identificar la variable recursiva**
2. **Reescribir la función como ecuaciones recursivas**
3. **Escribir el tipo de f y g**
4. **Definir la f**
5. **Definir la g o $\mathcal{G}$**

Veamos estos pasos con un ejemplo más complejo:

Sea $\Sigma = \{?, @, !, \%\}$

Y sea la función que queremos expresar como $R(f, \mathcal{G})$

$$F = \lambda xy\alpha_1\alpha[x + |\alpha_1| + |\alpha|_!]$$

Para este ejercicio vamos a asumir que tenemos demostrado que la suma y la cantidad de caracteres de una palabra son funciones Σ-PR

Paso 1: Identificar la variable recursiva

Tenemos que $F : \omega^2 \times \Sigma^{*2} \to \omega$

La variable recursiva es la última, la α. Y como F devuelve numéros, estamos en el caso **AN**

Paso 2: Reescribir la función como ecuaciones recursivas

$F(x, y, \alpha_1, \varepsilon) = x + |\alpha_1|$

$$F(x, y, \alpha_1, \alpha a) = \begin{cases} F(x, y, \alpha_1, \alpha) + 1 & \text{si } a =! \\ F(x, y, \alpha_1, \alpha) & \text{caso constrario} \end{cases}$$

Acá vemos que hay separación de casos. Siempre que hagamos una separación así en este paso, vamos a ver que aparecen múltiples definiciones al ver $\mathcal{G}$, pero esto lo veremos en el paso 5

Paso 3: Escribir el tipo de f y g

$f : \omega \times \omega \times \Sigma^* \to \omega$

$g_a : \omega \times \omega \times \omega \times \Sigma^* \times \Sigma^* \to \omega \quad$ con $a \in \Sigma$

Notar que en este caso el bloque fijo es $\text{BLK} = \omega \times \omega \times \Sigma^*$ y como estamos en el caso **AN** la declaración es $\omega_A \times \text{BLK} \times \Sigma_R^*$

Paso 4: Definir la f

Siguiendo el tipo de función y el valor que debe tener, obtenemos:

$$f(x, y, \alpha_1) = x + |\alpha_1|$$

Pero queremos declararla con funciones Σ-PR. Entonces queda:

$$f = \lambda ab[a + b] \circ \left[P_1^{2,1} \ , \quad \lambda\alpha[|\alpha|] \circ P_3^{2,1}\right]$$

Recordar que suponemos que ya probamos que $\lambda ab[a + b]$ y $\lambda\alpha[|\alpha|]$ son Σ-PR

Si bien esta declaración es compleja, se puede ver que efectivamente es:

$$\begin{aligned} f \ : \omega \times \omega \times \Sigma^* \quad &\to \omega \\ x, y, \alpha_1 \quad &\to x + |\alpha_1| \end{aligned} \tag{2.2}$$

Paso 5: Definir la g

Acá vemos la aparición de $\mathcal{G}$ en lugar de g. En este caso si debemos separar en casos:

$$\begin{aligned} \mathcal{G}_! \ : \omega \times \omega \times \omega \times \Sigma^* \times \Sigma^* \quad &\to \omega \\ A, x, y, \alpha_1, \alpha \quad &\to A + 1 = Suc(P_1^{3,2}) \end{aligned} \tag{2.3}$$

$$\begin{aligned} \mathcal{G}_a \ : \omega \times \omega \times \omega \times \Sigma^* \times \Sigma^* \quad &\to \omega \\ A, x, y, \alpha_1, \alpha \quad &\to A = P_1^{3,2} \end{aligned} \tag{2.4}$$

con $a \in \Sigma - \{!\}$

Con estas f y $\mathcal{G}$ tenemos que:

$$F = \lambda xy\alpha_1\alpha[x + |\alpha_1| + |\alpha|_!] = R(f, \mathcal{G})$$

2.3 Funciones Σ Recursivamente Primitivas

Def: **Funciones Σ-recursivamente primitivas**

Las funciones Σ-recursivamente primitivas será el conjunto PR^{Σ} tal que:
$\mathrm{PR}_0^{\Sigma} = \{Suc, Pred, C_0^{0,0}, C_{\varepsilon}^{0,0}\} \cup \{d_a \mid a \in \Sigma\} \cup \{p_j^{n,m} \mid 1 \leq j \leq n+m\}$
$A_k = \{f \circ [f_1, \ldots, f_r] \mid f, f_1, \ldots, f_r \in \mathrm{PR}_k^{\Sigma}, r \geq 1\}$
$B_k = \{R(f, \mathcal{G}) \mid R(f, \mathcal{G}) \text{ está definida } \wedge \{f\} \cup \{\mathcal{G}_a \mid a \in \Sigma\} \subseteq \mathrm{PR}_k^{\Sigma}\}$
$C_k = \{R(f, g) \mid R(f, g) \text{ está definida } \wedge f, g \in \mathrm{PR}_k^{\Sigma}\}$
$\mathrm{PR}_{k+1}^{\Sigma} = \mathrm{PR}_k^{\Sigma} \cup A_k \cup B_k \cup C_k$
$\mathrm{PR}^{\Sigma} = \bigcup_{k \geq 0} \mathrm{PR}_k^{\Sigma}$
Una función se llama Σ-recursiva primitiva (Σ-p.r.) si pertenece a PR^{Σ}

Algunas funciones que se demuestran en las guías que son Σ-PR:

- $\varnothing \in \mathrm{PR}^{\Sigma}$
- $\lambda xy[x + y] \in \mathrm{PR}^{\Sigma}$
- $\lambda xy[x.y] \in \mathrm{PR}^{\Sigma}$
- $\lambda xy[x!] \in \mathrm{PR}^{\Sigma}$
- $\lambda \alpha\beta[\alpha\beta] \in \mathrm{PR}^{\Sigma}$
- $\lambda \alpha[|\alpha|] \in \mathrm{PR}^{\Sigma}$
- $C_k^{n,m}, C_{\alpha}^{n,m} \in \mathrm{PR}^{\Sigma}\ \forall\ n, m, k \geq 0$ y $\alpha \in \Sigma$
- $\lambda xy[x^y] \in \mathrm{PR}^{\Sigma}$
- $\lambda t\alpha[\alpha^t] \in \mathrm{PR}^{\Sigma}$
- $\lambda xy[x \dot{-} y] \in \mathrm{PR}^{\Sigma}$
- $\lambda xy[\max(x, y)] \in \mathrm{PR}^{\Sigma}$
- $\lambda xy[x = y] \in \mathrm{PR}^{\Sigma}$
- $\lambda xy[x \leq y] \in \mathrm{PR}^{\Sigma}$
- $\lambda \alpha\beta[\alpha = \beta] \in \mathrm{PR}^{\Sigma}$

Def: **Operadores Lógicos**

Dados predicados $P : S \subseteq \omega^n \times \Sigma^{*m} \to \omega$ y $Q : S \subseteq \omega^n \times \Sigma^{*m} \to \omega$ definimos nuevos predicados que llamaremos: $(P \vee Q), (P \wedge Q)$ y $\neg P$ de la siguiente manera:

$$P \vee Q : S \to \omega$$
$$(\vec{x}, \vec{\alpha}) \to \begin{cases} 1 & P(\vec{x}, \vec{\alpha}) = 1 \text{ o } Q(\vec{x}, \vec{\alpha}) = 1 \\ 0 & \text{caso contrario} \end{cases} \tag{2.5}$$

$$P \wedge Q : S \to \omega$$
$$(\vec{x}, \vec{\alpha}) \to \begin{cases} 1 & P(\vec{x}, \vec{\alpha}) = 1 \text{ y } Q(\vec{x}, \vec{\alpha}) = 1 \\ 0 & \text{caso contrario} \end{cases} \tag{2.6}$$

$$\neg P : S \to \omega$$
$$(\vec{x}, \vec{\alpha}) \to \begin{cases} 1 & P(\vec{x}, \vec{\alpha}) = 0 \\ 0 & P(\vec{x}, \vec{\alpha}) = 1 \end{cases} \tag{2.7}$$

Lemma:
Si $P : S \subseteq \omega^n \times \Sigma^{*m} \to \omega$ y $Q : S \subseteq \omega^n \times \Sigma^{*m} \to \omega$ son predicados, entonces $(P \vee Q), (P \wedge Q)$ y $\neg P$ también lo son.

Def: **Conjuntos Σ-PR**
Es la **misma** definición que conjuntos Σ-e.e y Turing computables: Si su función característica $\chi_S^{\omega^n \times \Sigma^{*m}}$ lo es.

Lemma:
Sean $S_1, \ldots, S_n \subseteq \omega$ y $L_1, \ldots, L_m \subseteq \Sigma^*$ conjuntos no vacíos. Entonces:

$$S_1 \times \cdots \times S_n \times L_1 \times \cdots \times L_m \text{ es } \Sigma - PR \iff S_1, \ldots, S_n, L_1, \ldots, L_m \text{ son } \Sigma - PR$$

Def: **Función Restricción** $(f|_S)$
Dada una función f y un conjunto $S \subseteq D_f$, definimos $f|_S$ como:

$$f|_S = f \bigcap (S \times I_f)$$

y la llamamos la *restricción* de f al conjunto S

Notar que la función $f|_S$ es nada menos que:

$$\begin{aligned} f|_S : S &\to I_f \\ e &\to f(e) \end{aligned} \tag{2.8}$$

Lemma: **La restricción mantiene Σ-PR**
Si $f : D_f \subseteq \omega^n \times \Sigma^{*m} \to O$ es Σ-PR donde $O \in \{\omega, \Sigma^*\}$ y $S \subseteq D_f$ es Σ-PR entonces $f|_S$ es Σ-PR

Lemma: **'Extensión' de dominio**
Sean $O \in \{\omega, \Sigma^*\}$ y $n, m \in \omega$
Si $f : D_f \subseteq \omega^n \times \Sigma^{*m} \to O$ es Σ-PR entonces existe una función Σ-PR $\overline{f} : \omega^n \times \Sigma^{*m} \to O$ tal que:

$$f = \overline{f}|_{D_f}$$

Proposición importante:

Proposición:
Sea $S \subseteq \omega^n \times \Sigma^{*m}$. Se tiene que S es Σ-PR **si y solo sí** S es el dominio de alguna función Σ-PR

Lemma: **Funciones con división por casos**

Si tenemos funciones $f_1, \ldots, f_k$ tal que:
$f_i : D_{f_i} \to O$ para $O \in \{\omega, \Sigma^*\}$ y $1 \geq i \geq k$ tales que:
$D_{f_i} \cap D_{f_j} = \varnothing$ para $i \neq j$ Entonces

$$g = f_1 \bigcup \cdots \bigcup f_k$$

Es una función. En particular es la función:

$$\begin{aligned} g : \bigcap_{i=1}^{i=k} D_{f_i} &\to O \\ e &\to \begin{cases} f_1(e) & \text{si } e \in D_{f_1} \\ \ldots & \\ f_k(e) & \text{si } e \in D_{f_k} \end{cases} \end{aligned} \tag{2.9}$$

Lemma: **Conservación de Σ-PR en division por casos**

Sean $f_1, \ldots, f_k$ funciones Σ-PR tales que $f_i : D_{f_i} \to O$ para $O \in \{\omega, \Sigma^*\}$ y $1 \geq i \geq k$ tales que $D_{f_i} \cap D_{f_j} = \varnothing$ para $i \neq j$ (dominios disjuntos)
Entonces $f_1 \bigcup \cdots \bigcup f_k$ es Σ-PR

Def: **Sumatoria, productoria y concatenatoria de funciones Σ-PR**

Sea Σ un alfabeto finito. Sea $f : \omega \times S_1 \times \cdots \times S_n \times L_1 \times \cdots \times L_m \to \omega$ con $S_1, \ldots, S_n \subseteq \omega$ y $L_1, \ldots, L_n \subseteq \Sigma^*$
definimos:

$$\sum_{t=x}^{t=y} f(t, \vec{x}, \vec{\alpha}) \begin{cases} 0 & \text{si } x > y \\ f(x, \vec{x}, \vec{\alpha}) + f(x+1, \vec{x}, \vec{\alpha}) + \cdots + f(y, \vec{x}, \vec{\alpha}) & \text{si } x \leq y \end{cases}$$

$$\prod_{t=x}^{t=y} f(t, \vec{x}, \vec{\alpha}) \begin{cases} 1 & \text{si } x > y \\ f(x, \vec{x}, \vec{\alpha}) + f(x+1, \vec{x}, \vec{\alpha}) + \cdots + f(y, \vec{x}, \vec{\alpha}) & \text{si } x \leq y \end{cases}$$

Y cuando $f : \omega \times S_1 \times \cdots \times S_n \times L_1 \times \cdots \times L_m \to \Sigma^*$, definimos:

$$\subset_{t=x}^{t=y} f(t, \vec{x}, \vec{\alpha}) \begin{cases} \varepsilon & \text{si } x > y \\ f(x, \vec{x}, \vec{\alpha}) f(x+1, \vec{x}, \vec{\alpha}) \cdots f(y, \vec{x}, \vec{\alpha}) & \text{si } x \leq y \end{cases}$$

Lemma: **Lemma de la sumatoria**

Sea Σ un alfabeto finito.

(a) Si $f : \omega \times S_1 \times \cdots \times S_n \times L_1 \times \cdots \times L_m \to \omega$ es Σ-PR con $S_1, \ldots, S_n \subseteq \omega$ y $L_1, \ldots, L_n \subseteq \Sigma^*$ no vacíos, entonces las funciones:

$$\lambda xy \vec{x}\, \vec{\alpha} \left[\sum_{t=x}^{t=y} f(t, \vec{x}, \vec{\alpha}) \right]$$

y

$$\lambda xy \vec{x}\, \vec{\alpha} \left[\prod_{t=x}^{t=y} f(t, \vec{x}, \vec{\alpha}) \right]$$

son Σ-PR

(b) Si $f : \omega \times S_1 \times \cdots \times S_n \times L_1 \times \cdots \times L_m \to \Sigma^*$ es Σ-PR con $S_1, \ldots, S_n \subseteq \omega$ y $L_1, \ldots, L_n \subseteq \Sigma^*$ no vacíos, entonces la funcion:

$$\lambda xy \vec{x}\, \vec{\alpha} \left[\subset_{t=x}^{t=y} f(t, \vec{x}, \vec{\alpha}) \right]$$

es Σ-PR